Status and Directions for the PYRAMID Parallel Unstructured AMR Library

Charles D. Norton John Z. Lou Thomas A. Cwik
Jet Propulsion Laboratory
California Institute of Technology
MS 168-522, 4800 Oak Grove Drive
Pasadena, CA 91109-8099 USA
Charles.D.Norton@jpl.nasa.gov

Abstract

This is a status report on our progress with the development of PYRAMID, a Fortran 90/95-based library for parallel unstructured adaptive mesh refinement. The library has been designed to simplify the use of adaptive methods in computational science applications by introducing many advanced software engineering features. In this paper, design and performance issues are described concluding with a discussion of our future development plans.

1. Introduction

During the last 1.5 work years we have been actively developing a new kind of library for parallel unstructured adaptive mesh refinement called PYRAMID. While the software supports triangular and tetrahedral meshes, load balancing, mesh migration, partitioning, adaptive mesh quality control, and visualization, all in parallel, its object-based design in Fortran 90/95 and its ease of use have increased its appeal. In fact, ease of use concerns, as well as requirements specified by actual and potential users, drive library development. In this paper we will describe the current status of the library and new directions that expand on material presented at the Irregular 1998 workshop [5].

2. PYRAMID Library Overview

Parallel adaptive methods support the solution of complex problems using grid-based techniques. Software development for unstructured adaptive mesh refinement focuses on simplifying how the user interacts with the grid for problems that exhibit complex geometry. Research on how this is best achieved is on-going since meshes can be adaptively refined, partitioned, and load balanced using a variety of approaches.

Our library supports automatic mesh quality control ensuring that elements with poor aspect ratios are not created. This is achieved by actively redefining how an element is refined to prevent the creation of narrowly shaped triangles and tetrahedrons. The ParMetis graph partitioning software, from the University of Minnesota, is used with our mesh migration algorithms to load balance the adaptively refined mesh [4]. Our development in Fortran 90/95 allows for object-based design where a mesh can be created and manipulated using high-level library commands. A complete, minimal, PYRAMID program is shown in Figure 1.

```
program pyramid_example
use pyramid_module
implicit none
type (mesh), dimension(2) :: meshes
   call pamr_init()
   call pamr_create_incore(meshes(1), "mesh_data")
   call pamr_repartition(meshes(1))
   do i = 1, refinement_depth
      call pamr_error_est(meshes(1), meshes(2))
      call pamr_logical_amr(meshes(1))
      call pamr_repartition(meshes(1))
      call pamr_physical_amr(meshes(1), meshes(2))
   end do
   call pamr_visualize(meshes(2), "mesh.plt")
   call pamr_finalize(mpi_active = .true.)
end program pyramid_example
```

Figure 1. A minimal PYRAMID program.

Many of these routines accept Fortran 90/95 optional arguments that allow for additional control over the actions taken. When these options are not specified a reasonable default action is taken. Furthermore, for the most sophisticated routines, keyword arguments may optionally be specified to visually remind the user how various arguments are used.

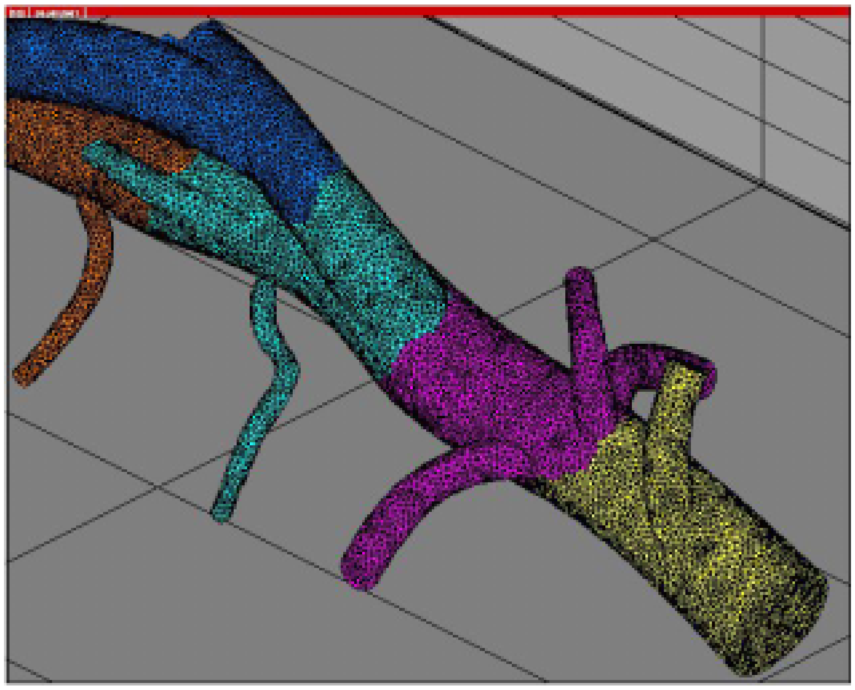


Figure 2. Artery mesh segment (courtesy of SCOREC, Rensselaer Polytechnic Institute).

The new features of Fortran 90/95 allow for complex data structures to be created, and used, while supporting encapsulation of related concepts in modules. Use of modules make the data and routines they contain available to program units and they can interact to provide increased functionality. Features such as smart pointers, generic interfaces, whole and array subset operations, and dynamic storage provide the modern software principles required for this work. This Fortran-based approach will simplify the union of adaptive meshing with existing Fortran-based solvers. This has resulted in a following within NASA, and elsewhere, regarding our progress with PYRAMID—this forms the primary purpose and focus of our article.

3. Features and New Directions

Demonstration versions of the PYRAMID library are accessible from the NASA HPCC Software Repository (http://ess.gsfc.nasa.gov/catalog.pl?rh=12). To date, the library has been used primarily on demonstration meshes to evaluate functionality and to test various features. On occasion, however, it has been applied to applications within JPL including adaption of multi-scale meshes for active device modeling [1].

Figure 2 shows a section of a tetrahedral finite element artery blood flow mesh generated by the Scientific Computation Research Group (SCOREC) at Rensselaer Polytechnic Institute [3] based on a geometry provided by Taylor et. al [7]. This mesh contains 1.1 million elements where the PYRAMID repartitioning, load balancing, and mesh migration are illustrated.

Another example, shown in Figure 3, demonstrates how a muzzle brake/shock tube mesh with 34,214 elements has been repartitioned and load balanced by PYRAMID. These meshes were produced by the program segment in Figure 1 where the refinement loop was not called. (Adaptive refinement will be discussed later.) The library simplifies a complex parallel operation into simple, easy to understand, operations.

One of the main problems with most adaptive mesh libraries is that accessing the mesh structure can be very awkward and confusing. Also, useful mathematical routines are not available as library commands and portability/performance issues across traditional supercomputers and clusters are not measured. Our current work has examined these issues and our new directions are aimed at increased functionality, such as parallel mesh generation support and other features.

3.1. PYRAMID Algorithms and Performance Enhancements

There are many techniques that PYRAMID employs on a software engineering and algorithm level to support unstructured adaptive meshing on triangular and tetrahedral meshes. We utilize the object-oriented aspects of For-

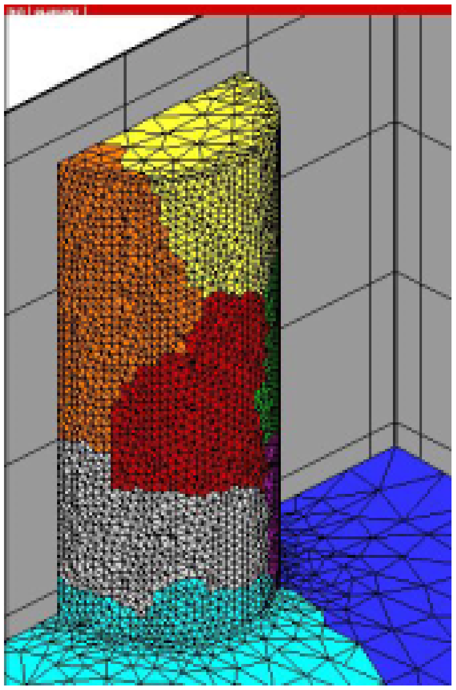


Figure 3. Muzzle brake/shock tube mesh (courtesy of SCOREC, Rensselaer Polytechnic Institute).

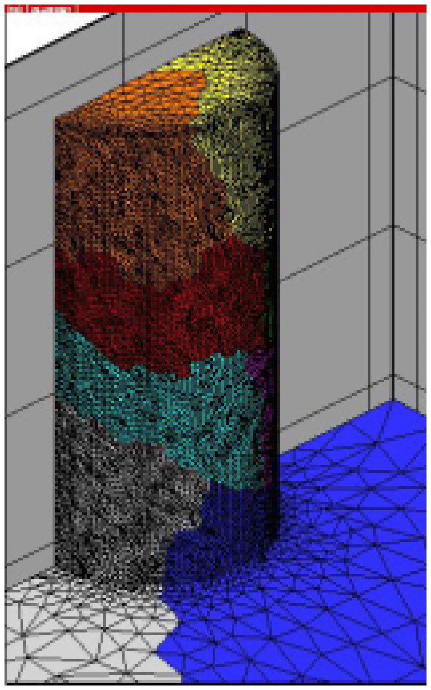


Figure 4. Refined Muzzle brake/shock tube mesh.

tran 90/95 [2] to define a mesh data structure that provides efficient access to all aspects of a mesh geometry in multiple dimensions. An automatic mesh quality control feature also ensures good element geometry, based on a green's refinement method, as adaptive refinement is applied. Aspects of our algorithm design are described, but more details are available elsewhere [6]. Every operation the library performs is a parallel operation.

Fortran 90/95 modules allow us to build software components that encapsulate various features of the data structure, such as connectivity and geometric boundary features, in a way that insulates such details from library users. Since module components can be specified as public and/or private our design uses information-hiding to protect and separate data structure internals from features a user requires. We use dynamic structures for memory management, but this can be done safely and efficiently since the proper technique can be selected based on the circumstance (pointers are never used when an allocatable array will work just as well).

MPI is used for message passing and all data structures managed by the library are distributed data structures. When adaptivity is applied a significant amount of work involves managing the distributed data structure to ensure that the mesh connectivity is efficiently specified. We employ heap sorting techniques for efficient mesh reconstruction during data mesh migration among distributed processors and for data access during adaptive refinement. This allows the library to quickly access any mesh component using binary search.

The adaptive refinement stage consists of two parts, a logical followed by a physical refinement operation. Refinement is specified by an error indicator which is defined geometrically for the examples in this paper. Logical refinement uses edge marking to specify how elements in the coarse mesh will be refined, but this stage does not actually create any elements. Based on this marking a refinement pattern for the element is determined and matched to a library of refinement patterns that are pre-defined. In addition, the elements are weighted based on these patterns and this information is provided to the load balancing and mesh migration components to compute a balanced partitioning for the set of elements that will be created in the physical refinement stage. Note, however, that users can always decide if repartitioning is needed.

The logical refinement also manages mesh quality control by detecting when poor aspect ratio elements will be created. If such elements may be introduced, based on examining when and how transitional elements are created, a refinement template will be applied that replaces these elements with new elements of higher quality. This preserves

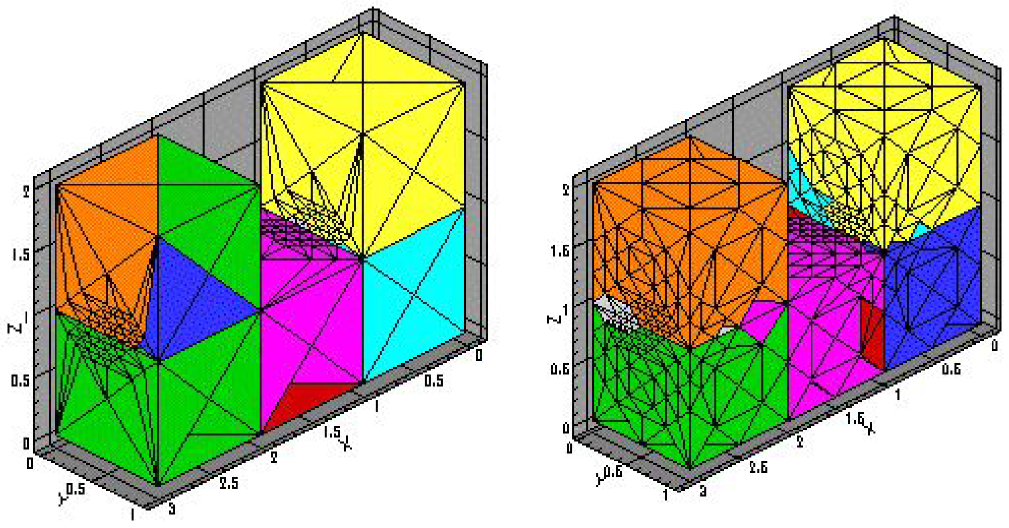


Figure 5. The effect on mesh quality with and without automatic quality control.

the structure of the mesh better than smoothing techniques at the expense of slightly more elements. An example of the importance of maintaining mesh quality during adaptive refinement is shown in Figure 5. The physical refinement stage creates the new elements based on the logical refinement specification. Of course, the coarse logical elements are migrated to new processors as needed before physical refinement is applied.

The logical refinement stage requires minimal communication to resolve refinement issues at partition boundaries. The physical stage is entirely local requiring no communication. This allows each of these stages to operate in a nearly embarrassingly parallel manner.

When applying repartitioning and mesh migration to the artery in Figure 2 we observed message passing race conditions that hindered performance significantly. Furthermore, certain mesh migration algorithms could potentially deadlock based on the availability of MPI internal message buffers. These subtle effects resulted from algorithms that sent and received messages as frequently as possible. The irregular nature of the communication imposed additional performance problems since message collisions on the cluster switch encouraged successive retransmissions. The communication structure for adaptive meshing problems is irregular in that the amount of data sent and received among processors is imbalanced, but it is largely predictable. These factors caused us to experience an average of about 1MB per second, on a 100BaseT full-duplex Ethernet connection, for data load balancing and mesh migration for the artery mesh. This was not exhibited in other mesh test cases.

We decided to introduce new algorithms for communication in the mesh migration stages. We developed a tree-based exchange algorithm that works for any number of processors. This algorithm is logarithmic, completely non-blocking and it can utilize full-duplex connections for machines that support this capability. It represents an extension of a pair-wise exchange utilizing techniques that don't require an actual communication tree to be physically constructed in memory. Additionally, the algorithm allows for data to be reduced during each iteration. Our original algorithms were very aggressive in conserving memory and this was a contributor to the performance problem we exhibited for the artery mesh. The new algorithm trades time for space, but the performance has been enhanced significantly. We now get a communication rate of nearly 11MB per second on the 100BaseT connection.

3.2. Object-Based Access to Mesh Structure

The sample program showed how a mesh object could be created and manipulated through member routines of the pyramid_module. This module acts like a C++ class since it defines the mesh components, the routines that manipulate the mesh, and it restricts the level of accessibility to the internal mesh data structure. Typically, a user must know the internal organization of the mesh to access information such as the coordinates of the first node of the second element, seen in Figure 6.

This is awkward particularly if the data is distributed, the mesh data structure is complex, and you don't know the processor on which this information is located. Figure 7 shows how our library commands simplify this process.

Notice how the use of keyword arguments allows either

```
type (mesh) :: this
real, dimension(3) :: xyz_pos
xyz_pos = &
        this%nodes(this%elements(2)%node_indx(1))%coord
```

Figure 6. Awkward data access scheme.

```
type (mesh) :: this
real, dimension(3) :: xyz_pos
real, dimension(3,4) :: all_pos
xyz_pos = pamr_element_coord(this, element_indx=2, &
        node_indx=1)
all_pos = pamr_element_coord(this, element_indx=2)
```

Figure 7. PYRAMID data access scheme.

a specific, or all, node coordinates to be returned. Keyword arguments make the specified values clear, and they can be variables although constants are shown in this example. The return variable must be conforming with the result of the call and the compiler can detect this. Many data access routines are included in PYRAMID.

Referring to terms defined on mesh components by user-defined names is also useful. We have added commands that allow one to define the number of terms on mesh components and to map user-defined names to these terms as shown in Figure 8.

```
type (mesh) :: this
real, pointer, save :: heat, mx, my, mz
call pamr_define_mesh_terms(this, &
        num_element_terms=20, num_face_terms=9,&
        num_edge_terms=4)
call pamr_map_mesh_terms(this, element_indx=10, &
        face_indx=3, edge_indx=2, a1=heat, a2=mx, &
        a3=my, a4=mz)
heat=10.5 ; mx=3.1 ; my=5.0 ; mz=12.4
```

Figure 8. User-defined mesh term naming.

In the example each element may contain 20 terms/variables each face has 9, and 4 are associated with each edge. The mapping allows the 4 variables heat, mx, my, and mz to be assigned to the indicated edge where assigning/reading these values can be done through the variables. Although constants are shown variables may be used. If the mapping only specifies an element then the variables are assigned to that element. If the element and face are specified then the indicated face is assigned the user-defined variables, and so on.

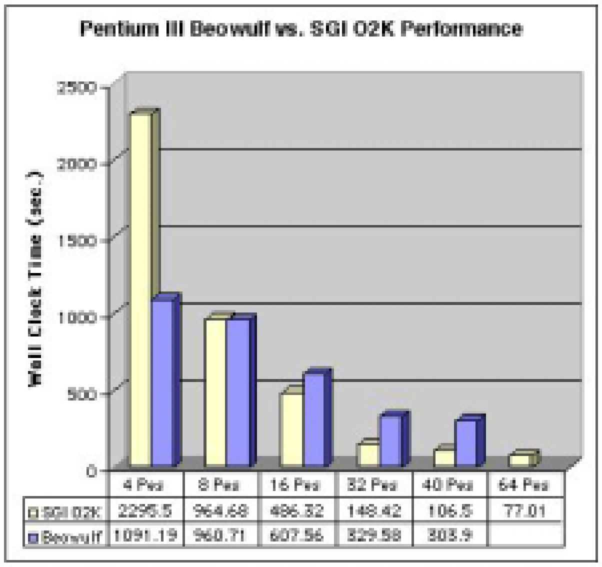
3.3. Mathematics Features

Numerous mathematical functions have been added to ensure that users get consistent results from the mesh data structure. The example in Figure 9 shows how a signed local unit normal basis for a specific face, or all faces, on an element may be returned. The basis consists of two unit normal vectors and the signed local unit normal for each face. The result format is indicated in the user manual where the values assigned must be conforming.

```
module pamr_interface_module
   function face_normalbasis_one(this, element_indx, &
   face_indx) result(n_basis)
      type (mesh), intent(in) :: this
      integer, intent(in) :: element_indx, face_indx
      real, dimension(3,3) :: n_basis
   end function

   function face_normalbasis_all(this, element_indx) &
   result(n_basis)
      type (mesh), intent(in) :: this
      integer, intent(in) :: element_indx
      real, dimension(3,3,4) :: n_basis
   end function
end module pamr_interface_module

module pyramid_module
use pamr_interface_module
   interface pamr_face_normalbasis
      module procedure face_normalbasis_one
      module procedure face_normalbasis_all
   end interface
   ! additional interfaces...
end module pyramid_module

program pyramid_example
use pyramid_module
integer :: the_elem, the_face
type (mesh) :: this
real, dimension(3,3,4) :: all_normals
real, dimension(3,4) :: face_normal
   the_elem = 100 ; the_face = 3
   face_normal = pamr_face_normalbasis(this, &
            element_indx=the_elem,face_indx=the_face)
   all_normals = pamr_face_normalbasis(this, &
            element_indx=the_elem)
end program pyramid_example
```

Figure 9. Mathematical mesh functions.

This returns to the user the [x, y, z] position of the 3 normal basis vectors for each face using a specified ordering. If



	4 Pes	8 Pes	16 Pes	32 Pes	40 Pes	64 Pes
▯ SGI O2K	2295.5	964.68	486.32	148.42	106.5	77.01
■ Beowulf	1091.19	960.71	607.56	329.58	303.9	

Figure 10. Performance after three refinement levels of the muzzle-brake mesh.

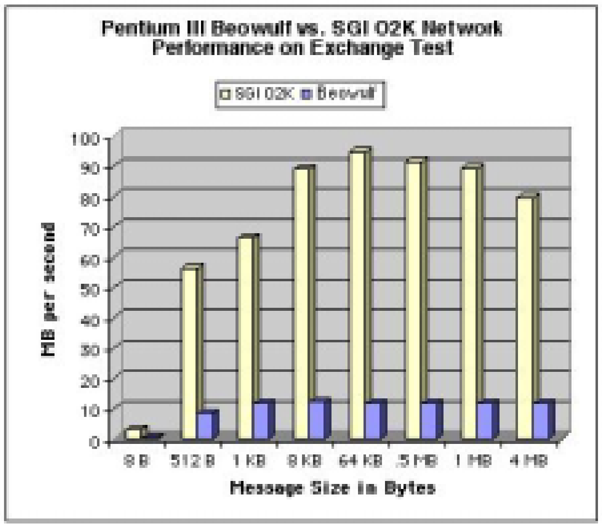


Figure 11. MBytes transmitted in pairwise processor data exchange between 2 PEs.

face_indx is specified this result can be returned for a specific face using the same command since Fortran 90/95 generic interfaces are supported, as shown in Figure 9. Other routines compute element and face centroids, volumes, areas and so on. Such features are added by user request.

3.4. Performance on Beowulf-Clusters, SGI Origin, and Cray T3E

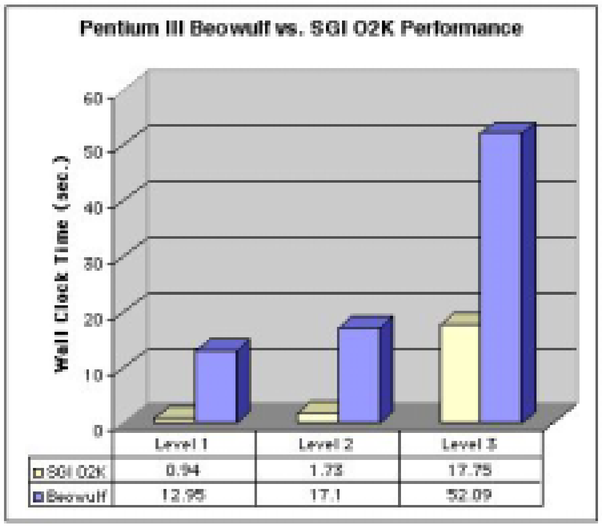
The Pyramid Library has been ported among Beowulf-Cluster systems, the SGI Origin, Cray T3E, and the IBM SP. Performance comparisons among the Beowulf cluster and SGI Origin are emphasized since the other systems were not available when these runs were performed. The Cray T3E results are based on runs performed before the system was decommissioned. The Beowulf cluster system consists of 21 dual-processor Pentium III 800MHz nodes with 100BaseT Ethernet for communication.

Figure 4 shows three levels of refinement for the tube section from Figure 3. The performance result in Figure 10 shows that the SGI O2K scales very well, even though the processor is much slower than the Pentium III for large input sizes. (Shared-memory message passing was not used for any of these systems.) The Beowulf-cluster competes well in comparison, but the network does hinder the performance of the fast processor for large problem sizes as seen in Figure 11. We expect to switch from the 100BaseT Ethernet to a Myrinet network shortly, so updated performance results will be available at our web site. The muzzle brake mesh contains 34,214 elements where 225,677, 350,847, and 1,264,443 elements are created after the 1st, 2nd, and 3rd refinements respectively.

On a small number of processors we see that the cluster can compete, and sometimes outperform, the SGI O2K, but as the number of processors increases the SGI gives better overall performance. It is interesting to examine how time is spent in the calculation. In Figure 12 we see that the SGI outperforms the Beowulf on 32 processors at each level of refinement, however, the percentage improvement is larger for the Beowulf cluster compared to the SGI O2K as the problem size grows during adaptive refinement.

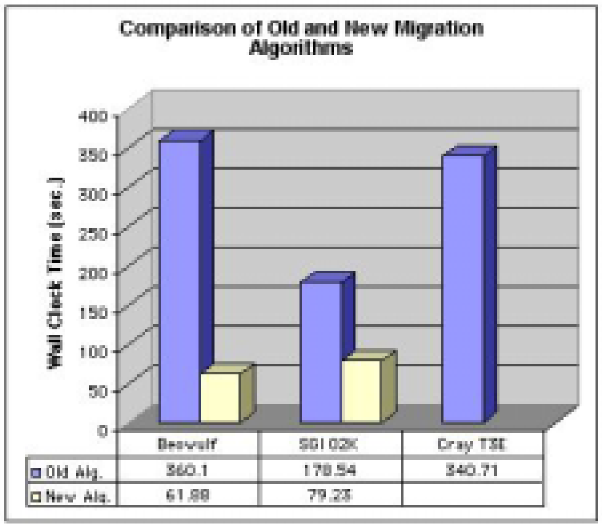
In Figure 13 we compare the performance of the old mesh migration algorithms on the earthquake mesh to the new exchange algorithm previously described. The improvement in performance for three levels of refinement on 8 processors is dramatic. In fact, the largest percentage gain is on the Beowulf cluster where an improvement in the communication scheme clearly benefited this architecture. The T3E was not available for comparison, but comparable improvements are expected.

Looking further into the time spent in adaptive refinement (logical and physical) and mesh migration in Figure 14 the effect of the network is clear for the Beowulf cluster. Remember that there is limited communication in the refinement stage which indicates that the Beowulf might outperform the SGI O2K in this stage if the network was faster. (This has been observed on smaller numbers of processors.) The migration times would also benefit from a faster network speed for the large messages sent in this stage. We estimate that the SGI has a 7 times network speed advantage on average during this stage.



	Level 1	Level 2	Level 3
□ SGI O2K	0.94	1.73	17.75
■ Beowulf	12.95	17.1	52.09

Figure 12. Earthquake mesh performance over three refinement levels on 32 PEs.



	Beowulf	SGI O2K	Cray T3E
■ Old Alg.	360.1	178.54	340.71
□ New Alg.	61.88	79.23	

Figure 13. Performance of algorithm modifications on the earthquake mesh for 8 PEs.

In these performance measurements the time to load and distribute the mesh data file among processors is included. The migration time includes mesh repartitioning, load balancing, reconstruction into the new mesh data structure and data migration among the processors.

In Figure 15 we see that the Beowulf cluster performs very well when compared to the SGI O2K for mesh repartitioning, dynamic load balancing, and migration of the artery mesh. Again, the network dominates the cluster for larger number of processors, but the per-node processor performance is most likely higher. The number of elements in this mesh is 1,103,018.

The communication performance enhancements have not yet been introduced into the refinement code, but this will be investigated. A number of geometric considerations were also introduced to reduce data communications. We feel that additional optimizations may be possible, but these are only introduced when the fundamental organization of the software can be preserved.

3.5. PYRAMID for Mesh Generation

Some users are interested in mesh generation on parallel computers using PYRAMID. In these instances an initial mesh is provided and Pyramid's refinement and load balancing algorithms uniformly increase the resolution of an initial coarse mesh as shown in Figure 16. The library package contains routines to convert a user's mesh into the PYRAMID mesh data format. There are also auxiliary routines that convert these meshes into distributed format for parallel I/O, often required for large meshes.

The library performs refinement based on a per-element error indicator, so we provide the means to apply unit error throughout the mesh for this capability.

3.6. Next Generation Features

Current work focuses on adding user-controllable boundary zones, interpolation methods among mesh levels, and defining straightforward ways to include error-estimation routines to drive the adaptive process. We may consider adding coarsening, but this will require careful planning for integration with the automatic mesh quality control features.

The demonstration version of the software is available from our web site (http://hpc.jpl.nasa.gov/APPS/AMR), but a number of the features described here are not included in that version. To ensure our design philosophy is preserved we add new functionality carefully. Anyone interested in contributing ideas is welcome to contact the authors.

4. Acknowledgment

We appreciate many helpful discussions with Viktor K. Decyk and Edward S. Vinyard. This work was developed at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

References

[1] T. A. Cwik, R. Coccioli, G. Wilkins, J. Z. Lou, and C. D. Norton. Multi-Scale Meshes for Finite Element and Finite Vol-

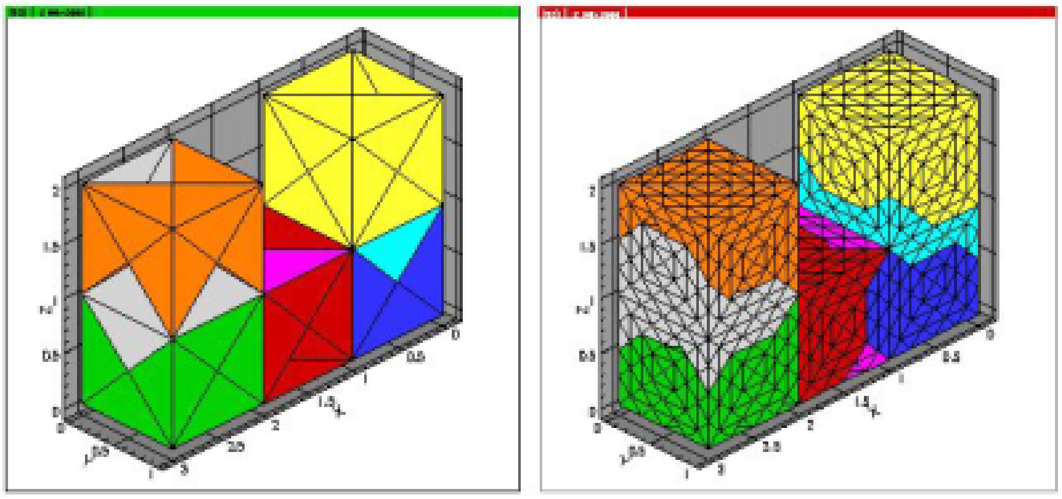
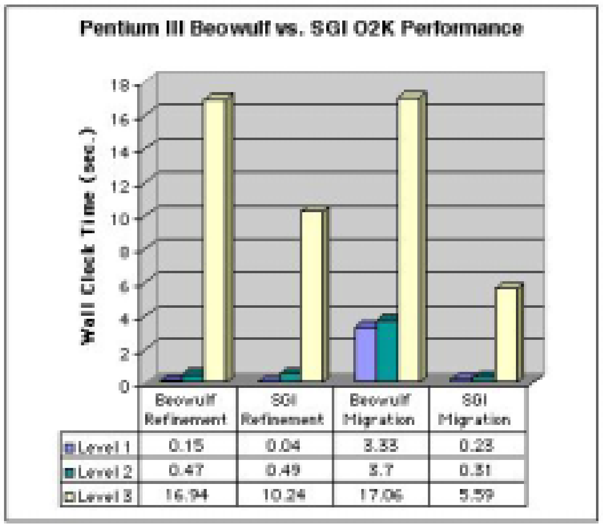
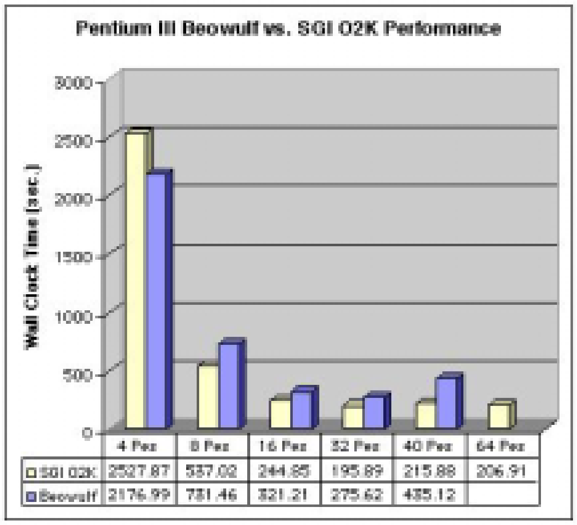


Figure 16. Use of mesh quality control feature for mesh generation.



	Beowulf Refinement	SGI Refinement	Beowulf Migration	SGI Migration
Level 1	0.15	0.04	3.33	0.23
Level 2	0.47	0.49	3.7	0.31
Level 3	16.94	10.24	17.06	5.59

Figure 14. Earthquake mesh refinement and migration over three adaptive levels on 32 PEs.



	4 Pes	8 Pes	16 Pes	32 Pes	40 Pes	64 Pes
SGI O2K	2527.87	537.02	244.85	195.89	215.88	206.91
Beowulf	2176.99	731.46	321.21	275.62	435.12	

Figure 15. Performance for mesh repartitioning and migration of the artery mesh.

ume Methods: Active Device and Guided-Wave Modeling. In *Proc. AP2000 Millennium Meeting*, Davos, Switzerland, April 2000.

[2] V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to Express C++ Concepts in Fortran 90. *Scientific Programming*, 6(4):363–390, Winter 1997. IOS Press.

[3] J. E. Flaherty and J. D. Teresco. Software for Parallel Adaptive Computation. In M. Deville and R. Owens, editors, *Proc. 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, 2000. IMACS. Paper 174–6.

[4] G. Karypis, K. Schloegel, and V. Kumar. ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Technical report, Dept. of Computer Science, U. Minnesota, 1997.

[5] J. Z. Lou, C. D. Norton, and T. Cwik. A Robust Parallel Adaptive Mesh Refinement Software Package for Unstructured Meshes. In *Proc. Fifth Intl. Symp. on Solving Irregularly Structured Problems in Parallel*, 1998.

[6] J. Z. Lou, C. D. Norton, and T. Cwik. The Development of a Parallel Adaptive Refinement Library for 2D and 3D Unstructured Meshes. In *Advanced Simulation Technologies Conference (ASTC)*, San Diego, CA, April 1999.

[7] C. A. Taylor, T. J. R. Hugues, and C. K. Zairns. Finite Element Modeling of Blood Flow in Arteries. To appear, *Comp. Meth. in Appl. Mech. and Engng.*, 1999.